AMATH 582 Homework 1

William Livernois (willll@uw.edu)

January 24, 2020

Abstract

The trajectory of a submarine was calculated and predicted from noisy acoustic data with an unknown characteristic frequency. Through Fourier analysis and filtering in MATLAB it was possible to determine the spatial frequency and track the location of the submarine in 3D space. Results were compared before and after filtering, showing a drastic improvement in measurement precision. Using this data, it was possible to predict the future location of the submarine.

1 Introduction and Overview

A submarine was detected in the southern Salish Sea area emitting an acoustic signal of unknown frequency. Acoustic measurements were taken near the submarine and processed to assign amplitude and phase data to a normalized cubic 20x20x20 region. Even after initial processing accounting for reflections and reference point timing, the resulting 3D arrays of data (taken at 30 min time intervals over a 24 hour period) remain noisy and unreliable. Combining Fourier analysis and simple filtering, the characteristic acoustic frequency was determined and the the signal to noise ratio was greatly improved. The resulting data was used to track and locate the submarine during the 24 hour period, with the potential to predict the future trajectory.

2 Theoretical Background

The Fourier Transform can be used to get spatial frequency information about the acoustic signal. Starting with a 3D complex function $f(x, y, z)$ the continuous Fourier Transform can be calculated as

$$\hat{f}(k_x, k_y, k_z) = \frac{1}{\sqrt{(2\pi)^3}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y, z) e^{i(k_x x + k_y y + k_z z)} \; dx \; dy \; dz \tag{1}$$

where the k values are in each dimension. Since our data, $U(x, y, z)$, is discrete and limited to the region $\{x, y, z\} \in [-L, L]$ we can change this to a discrete sum each each dimension to get a Fourier coefficient at each point on the k-mesh. By limiting the region in space and using a discrete mesh, with n points in each dimension, have upper and lower bounds to the wavenumber, k that are

$$\{k_x, k_y, k_z\} \in [\frac{-n\pi}{L}, \frac{n\pi}{L}] \tag{2}$$

Looking at the spatial frequency data, $\hat{U}_t(k_x, k_y, k_z)$, averaged over the 24 hour time interval, we can determine the characteristic frequency of the data by measuring the expected value in each dimension. To do this, we can normalize to transform it into a probability density function:

$$PDF = \frac{\left| \sum_{t=0}^{24} \hat{U}_t(k_x, k_y, k_z) \right|}{\iiint_V \left| \sum_{t=0}^{24} \hat{U}_t(k_x, k_y, k_z) \right| \; dk_x \; dk_y \; dk_z} \tag{3}$$

Since the data provided has noise in both time and space, a threshold must be applied to localize the probability function. This prevents the expected value from appearing around the origin due to the evenly spread noise across the whole mesh. After applying this cutoff, the PDF from Equation 3 can be used to determine the expected value of the wavenumber to be:

$$< k_i >= \iiint (PDF \cdot k_i) \; dk_x \; dk_y \; dk_z \text{ for } i \in \{x, y, z\} \tag{4}$$

With this value of k, a filter can be applied to the data at each time point. To do this, a discrete Fourier Transform will be appled to the data ($U_t(x, y, z) \to \hat{U}_t(k_x, k_y, k_z)$. A simple 3D gaussian filter can be used of the form

$$g(k_x, k_y, k_z) = \exp\left((k_x - < k_x >)^2 + (k_y - < k_y >)^2 + (k_z - < k_z >)^2\right) \tag{5}$$

scaling the original data at each point on the mesh. Finally an inverse Fourier Transform can be applied to the data ($\hat{U}_t(k_x, k_y, k_z) \times g(k_x, k_y, k_z) \to P_t(x, y, z)$). The location of the submarine can be determined by applying a cutoff to the data and finding the expected value of position using the same method as Equation 3:

$$\begin{aligned} < x > (t) &= \frac{\iiint x \cdot P_t(x, y, z) \; dx \; dy \; dz}{\iiint P_t(x, y, z) \; dx \; dy \; dz} \\ < y > (t) &= \frac{\iiint y \cdot P_t(x, y, z) \; dx \; dy \; dz}{\iiint P_t(x, y, z) \; dx \; dy \; dz} \\ < z > (t) &= \frac{\iiint z \cdot P_t(x, y, z) \; dx \; dy \; dz}{\iiint P_t(x, y, z) \; dx \; dy \; dz} \end{aligned} \tag{6}$$

Combining the filtering with integrating ensures that the maximum amount of information retained from the original data. This method assumes that one dominant wavenumber vector was present in the data (not changing with time) and assumes that the noise is random, using integration to smooth out local fluctuations in amplitude.

3 Algorithm Implementation and Development

The Fast Fourier Transform (FFT) algorithm was used to take the discrete Fourier Transform of the submarine data at each time. point. This algorithm used the Cooley-Tukey method that operates in $O(n \log n)$ runtime[1]. This method works by recursively dividing the series into two sections, which means that the mesh size is restricted to 2^n, although padding methods can be used for non-standard sizes. The implementation of the FFT in MATLAB leads to two behaviors: the resulting Fourier modes start at the zero-frequency (rather than being centered at 0) and the values alternate sign. For this reason, the k-space mesh and vectors are shifted (while applying the scaling factor) and only amplitudes were considered for cutoffs and filtering. For processing our data set a filter was built and applied to the FFT of the spatial data to find the submarine position at each time point. The center point of the filter and position calculation both required an expectation value calculation that generated a PDF and integrating. For this calculation, a function `ExpVal` was created that would input the operator (k_i or x_i) matrix, the probability data, and a cutoff threshold for localization of the PDF. This function worked as follows:

Algorithm 1: ExpVal Algorithm

```
Rescale probability data, $F$, to be between 0 and 1
for matrix element in $F$ do
    if matrix element < threshold then
        Set matrix element to zero
    end if
end for
$PDF = F / \left(\iiint F \; d^3r\right)$
return $\iiint PDF \times \text{Operator} \; d^3r$
```

The volume integral was implemented using the discrete trapezoidal method along each axis and the position and space operators were generated using the `meshgrid` function (described in Appendix A). To filter the spatial data at each time point, the 3D Gaussian filter matrix was calculated using the generated $\{k_x, k_y, k_z\}$ operator matrices.

As a final step, a helical function was fit to the data using the MATLAB curve fitting toolbox. These functions were of the form

$$\begin{aligned} x(t) &= A_x \sin[(B_x(t - C_x)] + D_x \\ x(t) &= A_y \sin[(B_y(t - C_y)] + D_y \\ z(t) &= A_z t + B_z \end{aligned} \tag{7}$$

with fit coefficients A, B, C, and D. These coefficients can be used to interpolate data or predict the submarine location at a future time.

4 Computational Results

The characteristic spatial frequency of the submarine acoustic data was found to be the k-vector $(5.43, -6.91, 2.20)$ with units of radians per length unit. The submarine location was determined at each time point and a plot of the trajectory can be seen in Figure 1 (left) below. The impact of the Gaussian filter can be clearly seen in this figure, especially for calculation of velocity between time frames. In addition, the helical trajectory is quite clear looking at the x, y, and z position data as a function of time as shown in Figure 1 (right). Based on this fit, the coordinates of the submarine location can be interpolated and extrapolated with the following functions:

$$\begin{aligned} x(t) &= -5.00 \sin[0.18 * (t - 11.29)] - 1.73 \\ y(t) &= 6.11 \sin[0.12 * (t + 0.15)] - 0.12 \\ z(t) &= 0.60 * t + -8.00 \end{aligned} \tag{8}$$

The average speed of the submarine was found to be a relatively constant 1 length unit per hour, with a variance of about 10% due to the minimal noise still present in the filtered output.

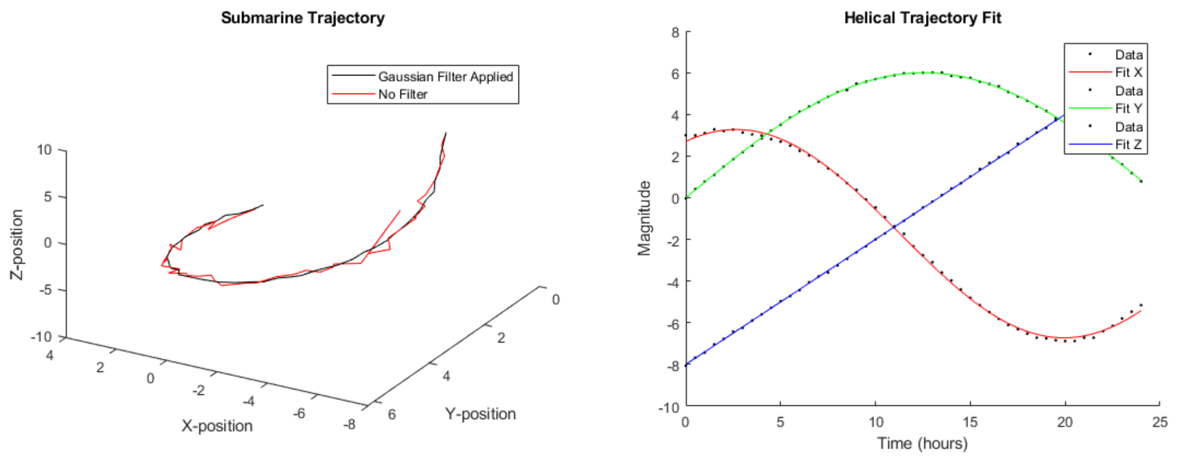

Figure 1: (left) A plot of the trajectory of the submarine plotted with and without the Gaussian filter applied and (right) a plot of the x, y, and z trajectory over time with a helix equation fit applied

5 Summary and Conclusions

For this specific case, the use of averaging and filtering in Fourier space was quite successful for determination of the characteristic spatial frequency and then the submarine trajectory. In addition, the use of integration methods (weighting a matrix as a probability density function) improved tracking accuracy by smoothing out any fluctuations that would occur at a threshold boundary. The calculated trajectory was smooth allowing for precise fitting of the data to a simple function.

For a more complicated system, the assumptions made here could lead to issues with data processing. If there were more than one characteristic frequency, multiple filters would need to be used on the data (rather than just at one k-point). In addition, the use of averaging to determine the characteristic spatial frequency would not work if the characteristic frequency changed as a function of time. One method that could be used to remove noise at each time step would be to use a window-function type method such as the Gabor Transform or a Wavelet transform. Fitting a trend to the varying frequencies it would be possible to filter the data at each time point and reproduce a precise trajectory.

References

[1] James W. Cooley and John W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series". In: *Mathematics of Computation* 19.90 (1965), pp. 297–301. ISSN: 00255718, 10886842. URL: `http://www.jstor.org/stable/2003354`.

Appendix A MATLAB Functions

The MATLAB functions used for these calculations are listed below:

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[X,Y, Z] = meshgrid(x,y,z)` returns 3-D matrices with coordinates that span the values contained in the vectors `x`, `y`, and `z`.
- `Y = fftshift(X)` shifts the Fourier Transform X along each dimension, moving the zero-frequency to the center.
- `Y = reshape(A, n1, n2, n3)` reshapes the `A` matrix to be an n1xn2xn3 matrix.
- `Q = trapz(x,f)` integrates f with respect to the coordinates x using the trapezoidal method
- `Fx = gradient(F, h)` takes the gradient of vector `F` with time step `h`
- `f = fit(t,y,fitType,fitOptions)` fits the function `y` to the function defined by the `fitType` object
- `fitType = fittype(expression,'independent',var)` generates a `fitType` object based on the MATLAB `expression` and given independent variable `var`

Functions used for basic arithmetic or visualization are not included above, but the important figures and numerical results are included in the Computational Results section.

Appendix B MATLAB Code

Only one file, HW1.m, was used for the calculations and visualizations and is shown below:

```
clear all; close all; clc

load subdata.mat

%Build space and wavenumber grids
L=10;
n=64;
x2 = linspace(-L, L, n+1); x=x2(1:n);
y = x;
z = x;
k = (2*pi/(2*L))*[0:(n/2 - 1) -n/2:-1]; ks = fftshift(k);
[X,Y,Z] = meshgrid(x,y,z);
[Kx, Ky, Kz] = meshgrid(ks, ks, ks);
Uk = zeros(n,n,n);

%Extract timestep data and average FFT results
for j=1:49
    Un(:,:,:)=reshape(subdata(:,j),n,n,n);
    M = max(abs(Un),[],'all');
    Uk = Uk + fftn(Un);
    % DEBUGGING PLOTS:
%     clf
%     isosurface(X,Y,Z,abs(Un)/M,0.5);
%     axis([-20 20 -20 20 -20 20]), grid on, drawnow
%     isosurface(Kx,Ky,Kz,abs(Uks)/max(abs(Uks(:))),0.5);
%     axis([-10 10 -10 10 -10 10]), grid on, drawnow
%     pause(0.2)
end

%Shift and scale FFT data
Uks = fftshift(Uk);
aveUks = abs(Uks)/max(abs(Uks(:)));

% DEBUGGING PLOTS:
% isosurface(Kx,Ky,Kz,aveUks,0.9);
% axis([-10 10 -10 10 -10 10]), grid on, drawnow

% Uks2 = Uks(:,:,39);
% PDF2 = abs(Uks2).^2/trapz(ks, trapz(ks, abs(Uks2).^2));
% PDF2 = PDF2.*(PDF2>0.01);
% PDF2 = PDF2/trapz(ks, trapz(ks, PDF2));
% Kx2 = X(:,:,1);
% Ky2 = Y(:,:,1);
% Uks2p = abs(Uks2)/max(abs(Uks2(:)));
% figure, contourf(Kx2,Ky2,PDF2)
%
% trapz(ks, trapz(ks, PDF2))
%
% kx2 = trapz(ks, trapz(ks, PDF2.*Kx2))
% ky2 = trapz(ks, trapz(ks, PDF2.*Ky2))
% Threshold for localization of operators
cutoff = 0.9;

% Find mean values of kx using averaged frequency amplitudes
kx = ExpVal(ks, aveUks, Kx, cutoff)
ky = ExpVal(ks, abs(Uks).^2, Ky, cutoff)
kz = ExpVal(ks, abs(Uks).^2, Kz, cutoff)

%A simple gaussian filter to apply in frequency domain
filter = ifftshift(exp(-0.1*((Kx - kx).^2 + (Ky - ky).^2 +(Kz - kz).^2)));
%isosurface(Kx,Ky,Kz,filter,0.9);

%Extract location at each time point using filtered data
subLoc = zeros(49, 3);
subLocNoF = zeros(49, 3);
for j=1:49
    Un(:,:,:)=reshape(subdata(:,j),n,n,n);
    Uf = ifftn(filter.*fftn(Un));
    Ufn = abs(Uf)/max(abs(Uf(:)));
    xLoc = ExpVal(x, Ufn, X, cutoff);
    yLoc = ExpVal(y, Ufn, Y, cutoff);
    zLoc = ExpVal(z, Ufn, Z, cutoff);
    subLoc(j, :) = [xLoc yLoc zLoc];
    xLocNoF = ExpVal(x, Un, X, cutoff);
    yLocNoF = ExpVal(y, Un, Y, cutoff);
    zLocNoF = ExpVal(z, Un, Z, cutoff);
    subLocNoF(j, :) = [xLocNoF yLocNoF zLocNoF];
end

%Plot submarine trajectory, compare to unfiltered trajectorynb
figure;
plot3(subLoc(:, 1), subLoc(:, 2), subLoc(:, 3), '-k', subLocNoF(:, 1), subLocNoF(:, 2), subLocNoF(:, 3)
title('Submarine Trajectory')
legend('Gaussian Filter Applied', 'No Filter', 'Location', 'best')
xlabel('X-position')
ylabel('Y-position')
zlabel('Z-position')

%Get velocity information
subVelVec = [gradient(subLoc(:, 1), 0.5), gradient(subLoc(:,2), 0.5), gradient(subLoc(:,3), 0.5)];
subSpeed = sqrt((subVelVec(:,1).^2)+(subVelVec(:,2).^2)+(subVelVec(:,3).^2));
%figure, plot(subSpeed);
aveSpeed = mean(subSpeed)
stdSpeed = std(subSpeed)

%Fit Data to Helical functions
t=linspace(0,24,49);
f = fittype('a*sin(d*(x+b))+c', 'independent', 'x');
fx = fit(t', subLoc(:, 1), f, 'StartPoint', [4 -12 -2 0.24]);
fy = fit(t', subLoc(:, 2), f, 'StartPoint', [3 -5 3 0.1817]);
fz = fit(t', subLoc(:, 3), 'poly1');

%Plot Fit Results
figure;
```

```
hold on;
plot(fx, '-r', t', subLoc(:, 1), '.k')
plot(fy, '-g', t', subLoc(:, 2), '.k')
plot(fz, '-b', t', subLoc(:, 3), '.k')
ylabel('Magnitude')
legend('Data', 'Fit X', 'Data', 'Fit Y', 'Data', 'Fit Z')
xlabel('Time (hours)')
title('Helical Trajectory Fit')
hold off

%Print Functions
disp('Helix Fit Functions:')
disp('--------------------')
fprintf('x(t) = %5.2f sin[%5.2f * (t + %5.2f)] + %5.2f\n', fx.a, fx.d, fx.b, fx.c);
fprintf('y(t) = %5.2f sin[%5.2f * (t + %5.2f)] + %5.2f\n', fy.a, fy.d, fy.b, fy.c);
fprintf('z(t) = %5.2f * t + %5.2f\n', fz.p1, fz.p2);

%Function for finding 3D integral using trapz method
function int = Int3(x_, F)
    int = trapz(x_, trapz(x_, trapz(x_, F)));
end

%Find expected value of operator M using F as a PDF, F is filtered with a
%simple threshold thres to localize the generated PDF
function val = ExpVal(x_, F,  M, thres)
    F = abs(F)/max(abs(F(:)));
    F = F.*(F>thres);
    PDF = F/Int3(x_, F);
    val = Int3(x_, PDF.*M);
end
```